IN THE CLAIMS:

36. (Currently Amended) A method for an original operating system (a host OS) in a mobile device that supports a memory protection mechanism to run another operating system (a guest OS) within the same memory space of said host OS while preserving the current state of said host OS in memory throughout the execution of said guest OS, comprising the steps of:

said mobile device running said host OS;

said host OS starting said guest OS through a launcher;

said launcher going [[go]] through said memory protection mechanism to mark memory blocks currently used by said host OS as protected from said guest OS;

said launcher launching said guest OS;

said guest OS running, accessing only memory blocks that have not been marked as protected so that said memory blocks marked as protected are preserved throughout the execution of said guest OS;

said guest OS finishing ~~finish~~ running through an exit-code;

said exit-code restoring ~~restores~~ the state of said host OS by reverting said protected memory blocks; and

said host OS resuming its operation.

37. (Currently Amended) The method of claim 36, wherein said memory protection mechanism is achieved through a memory management unit (MMU) that allows or disallows ~~allow or disallow~~ a program to access particular memory addresses.

38. (Currently Amended) The method of claim 36, wherein said launcher going [[go]] through said memory protection mechanism to mark memory blocks currently used by said host OS as protected from said guest OS further comprises the ~~comprising~~ steps of:

moving memory blocks currently used by said host OS whose address range is needed by said guest OS to a free memory location in the mobile device; and

marking said free memory blocks as protected;

and wherein restoring ~~restores~~ the state of said host OS by reverting said protected memory blocks further comprises the ~~comprising~~ steps of:

restoring said free memory blocks to said memory blocks whose address range is needed by said guest OS; and

reverting said free memory blocks from protected.

39. (Currently Amended) The method of claim 36, further comprising starting ~~to start~~ a second guest OS from said guest OS, ~~further~~ wherein starting the second guest OS comprises ~~comprising~~ the steps of:

said guest OS acting as a host OS to the second guest OS;

said second guest OS acting as a guest OS; and

repeating the ~~Repeating~~ steps in claim 36 to start said second guest OS from said guest OS.

40. (Currently Amended) A method for an original operating system (a host OS) of a mobile device to start another operating system (a guest OS) while keeping the running state and data of said host OS in memory throughout the execution of said guest OS, comprising the steps of:

said mobile device running said host OS;

said host OS starting said guest OS through a launcher;

said launcher moving memory blocks in lower address space that are currently used by said host OS to free memory blocks in upper address space, and eventually preserving current state and data of said host OS to the upper address space through the following steps:

for each of said memory blocks in lower address space currently used by said host OS, finding ~~find~~ a free memory block in the upper address space; and

moving ~~move~~ said used memory block in lower address space to said free memory block in upper address space;

said launcher identifying a memory address location where memory addresses ~~memories~~ above said location contains host OS data and memory addresses below said memory address location are ~~to be~~ free to be used ~~use~~ by said guest OS;

said launcher launching said guest OS[[,]] by passing said memory address location as a reduced memory size to said guest OS;

said guest OS running in said reduced in size ~~reduce-sized~~ memory space and leaving ~~leave~~ memory space higher than said memory location untouched;

said guest OS finishing ~~finish~~ running through an exit-code;

said exit-code restoring ~~restores~~ the state and data of said host OS by reverting each of said memory blocks in said lower address space from each of said free memory blocks in said upper address space; and

said host OS resuming operation.

41. (Currently Amended) The method of claim 40, wherein said launcher launching said guest OS[[,]] by passing memory address location as a reduced memory size to said guest OS[[;]] further comprises ~~comprising~~:

a memory detection module of said guest OS that uses said reduced in size ~~reduce-sized~~ memory space as the total available memory of the mobile device instead of the real memory size of the mobile device.

42. (Currently Amended) The method of claim 40, wherein said launcher launching said guest OS[[,]] by passing memory address location as a reduced memory size to said guest OS[[;]] further comprises ~~comprising~~:

said launcher modifying a system register in said mobile device to report said reduced memory size as total available memory ~~memories~~ to said guest OS.

43. (Currently Amended) The method of claim 40, wherein memories of said mobile device are divided into multiple memory banks, and said launcher

launching said guest OS[[,]] by passing memory address location as a reduced memory size to said guest OS[[;]] further comprises ~~comprising~~:

generating ~~Generating~~ said reduced memory size by disabling one or more of said memory banks to make them no longer available to said guest OS.

44. (Currently Amended) The method of claim 40, wherein input output (IO) states[[,]] and registers of said mobile device within said host OS are preserved and later restored into said free memory block in upper address space.

45. (Currently Amended) A method for an original operating system (a host OS) of a mobile device to start another operating system (a guest OS) within the same memory space of said host OS while keeping the running state of said host OS throughout the execution of said guest OS in place, comprising the steps of:

said mobile device running said host OS;

said host OS starting said guest OS through a launcher;

said launcher launching said guest OS, passing a list of memory addresses of currently used memory of said host OS to a memory device driver in said guest OS during initialization of said guest OS;

said memory device driver of said guest OS claiming said list of memory addresses and keeping them from being modified by any other part of said guest OS during the execution of said guest OS;

said guest OS running in the same memory space of said host OS, with memories used by said host OS being claimed and protected by ~~under~~ said memory device driver;

said guest OS finishing ~~finish~~ running through an exit-code;

said exit-code restoring ~~restores~~ the state of said host OS by releasing said list of memory addresses from said device driver; and

said host OS resuming operation.

46. (Currently Amended) The method of claim 45, wherein said list of memory addresses used by host OS includes ~~include~~ memory addresses only in dynamic runtime memory of said host OS and not memories.

47. (Currently Amended) The method of claim 45, wherein said list of memory addresses used by host OS also includes ~~include~~ the current input output (IO) states and current registers of the mobile device.

48. (Canceled)

49. (Canceled)

50. (Currently Amended) A method of packaging an image of a guest OS inside a native application of a host OS to allow in-place execution of the said guest OS image in order to reduce memory usage where the format of said native application contains multiple un-continuous data chunks in memory space, comprising the steps of:

compiling an image of said guest OS into multiple code segments;

appending each of said multiple code segments with a jump table containing multiple jump addresses to be used to point to others of the ~~the other~~ code segments;

for each of a plurality of inter-segment jump instructions in each of said code segments, linking each of said plurality of inter-segment jump instructions to point to an entry of said jump table;

preparing a native application for said host OS with a startup code and multiple data chunks;

each of said multiple data chunks wrapping each of said code segments of said guest OS plus each of said jump tables;

said host OS launching said guest OS by running said native application;

said startup code in said native application looping through entries of each of said jump table ~~entries~~ to fill in the current memory address of each corresponding code segment ~~segments~~;

each of said code segments ~~segment~~ executing a jump instruction to invoke codes in the other ~~another~~ code segments ~~segment~~;

said executed jump instruction getting a real address of said other ~~another~~ code segment from said jump tables;

said executed jump instruction successfully jumping to the other code segments ~~segment~~ wrapped by a corresponding ~~said another~~ data chunk; and

wherein said image of guest OS executes ~~now executing~~ in place without the need to extract all the code segments from said native application and rearrange them in a continuous memory location.

51. (Currently Amended) The method of claim 50, wherein said native application in said host OS is a database file with multiple data records or a Palm PDB file with multiple Palm DB records.

52. (Currently Amended) The method of claim 50, wherein said native application in said host OS is a Palm PDB file with multiple Palm DB records and said host OS is a Palm OS.

53. (Currently Amended) The method of claim 50, wherein said image of said ~~the~~ guest OS code requires one sequential memory address, said native application of said host OS contains multiple data chunks that have ~~has~~ a maximum chunk size,[[;]] and compiling an image of said guest OS into multiple code segments further comprises ~~comprising~~ the steps of:

compiling said image of said guest OS into one code segment with sequential memory addresses ~~address~~;

re-arranging instructions in said image of said guest OS to reserve spaces to be used for said jump tables on each of a plurality of ~~said~~ chunk-size boundaries by inserting jump instructions to bypass ~~by pass~~ those spaces; and

splitting said one code segment into multiple code segments on each of said plurality of chunk-size ~~said~~ ~~chunk size~~ boundaries.[[;]]

54. (Currently Amended) The method of claim 50, wherein the size of said guest OS image exceeds the maximum size limit of said native application of said host OS, further comprising the steps of:

splitting said image of said guest OS into multiple pieces;

compiling each of said multiple pieces ~~piece~~ of said image of said guest OS into multiple code segments; and

packaging each of said multiple pieces ~~piece~~ of said image of guest OS into a native application of said host OS.[[;]]